DLIT: A Scalable Distributed Ledger for IoT Data

Niya, Sina Rafati ; Beckmann, Raphael ; Stiller, Burkhard

DLIT: A Scalable Distributed Ledger for IoT Data

Sina Rafati Niya, Raphael Beckmann, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH
Binzmühlestrasse 14, CH—8050 Zürich, Switzerland
Emails: [rafati|stiller@ifi.uzh.ch], raphael.beckmann@uzh.ch

***Abstract*—The integration of the Internet-of-Things (IoT) and Blockchain (BC) for strong trust and decentralization shows potentials in use cases, such as supply chain tracing, smart cities, and health care. As a great number of IoT devices interacting in such cases, it is crucial to provide scalable and secure mechanisms for IoT data persistence within BCs. In this regard, sharding mechanisms have been employed to enhance the scalability of BCs. However, disconnections and delays of a BC's distributed network can cause concerns for inter-shard and inter-miner synchronizations, eventually preventing the BC from reaching a high throughput. Thus, this work develops an IoT-oriented permissioned BC, which covers via a scalable Distributed Ledger (DL) a novel sharding mechanism for unstable distributed networks. Therefore, DLIT (Distributed Ledger for IoT Data) offers a novel two-layered transaction distribution, validation, and inter-shard synchronization, combined with authentication and verification mechanisms in support of a viable security level.**

***Index Terms*—Blockchain, Distributed Ledger, Internet-of-Things, Proof-of-Stake, Byzantine Fault Tolerance, Sharding.**

I. Introduction

Blockchains (BC) support distributed data storage in an immutable manner. This became visible during the last decade by the emergence of cryptocurrencies, such as Bitcoin, which paved the path for a strong evolution of trusted, decentralized, and disintermediated Information Technology (IT). Besides early cryptocurrencies, different use cases evolved around BCs, out of which IoT plays an important role. IoT especially focuses natively on a secure and decentralized data collection, monitoring, analytics, and storage due to its decentralization of IoT devices. A certain level of trust in IoT data is required to prepare and take control decisions [15]. Hence, examples of IoT-driven and BC-based use cases developed by time comprise supply chain tracing [11], smart cities [13], pollution monitoring [12], agriculture, and healthcare.

Further, BCs have evolved to a better scalability via new consensus mechanisms, such as Proof-of-Stake (PoS), Proof-of-Authority (PoA), or Byzantine Fault Tolerance (BFT) [5]. However, a combination of scalable and secure BCs has not been reached yet. Thus, private permissioned BCs (called Distributed Ledgers, DL) have gained interest, where a consortium of nodes is in control of managing the chain. DLs trade off scalability and security with centralization. To enhance the scalability of DLs, "sharding" mechanisms shown potential [17], which leads to an improvement in scalability of a DL with *(a)* an increasing number of miners added per shard and *(b)* by increasing the number of shards. However, inter-shard communications is required to synchronize miners, which causes networking problems to be exposed to delays and packet loss [1].

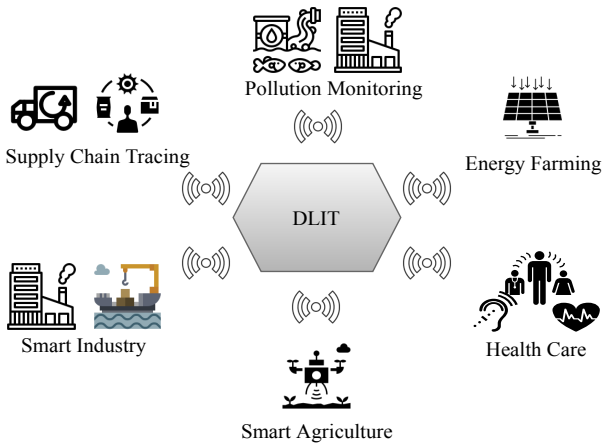

Fig. 1. IoT and Blockchain Integrated Use Cases

A second approach to foster DL scalability [14] suggests to aggregate Transactions (Tx) on chain. Tx aggregation controls and moderates the DL growth in size by combining multiple *Tx*s into a new *Tx*, reducing header overhead. Such a storage optimization approach was not yet used in sharding approaches [17]. Thus, to limit the impact of networking problems and eventually to reach good scalability (measured in Transaction validation rate Per Second, TPS), DLIT (Distributed Ledger for IoT Data) as proposed integrates these two worlds and enables a novel sharded Proof-of-Stake (PoS) consensus mechanism by applying Tx aggregation on validated Txs.

DLIT designs a DL with a two-layer consensus mechanism by combining Byzantine Fault Tolerance (BFT) and PoS. DLIT's sharding mechanism removes inter-shard synchronization requirements for state transitions. Thus, in unreliable networking conditions, the reduced number of inter-miner communications can be less time-demanding than a regular sharding approach. In real world, as shown in Fig. 1, different IoT use cases can generate high volume of *Tx*s to be stored in a DL. To adhere the security demands of such a high scale, DLIT reaches high security for storing IoT data by enabling slashing *i.e.,* fining mechanism on malicious nodes. Block mining is verified against the state transitions and assigned *Tx*s. DLIT offers a novel design in which validation processes are separated from the Tx assignment. Consortium members are selected randomly and perform with limited authorities.

The remainder of this paper is organized as follows. Section II formulates major concerns observed in sharding mechanisms and introduces related work. While the design and implementation of DLIT are presented in Section III, key evaluation results of experiments performed with the prototype implemented are presented in Section IV, followed by the summary Section V.

II. Sharding Concerns and Related Work

The basic idea of sharding originates from databases, where sharding denotes the horizontal partitioning of a database among multiple physical data stores [7] such that they can be processed concurrently [4]. A basic DL sharding design partitions the global *Tx mempool* (a memory to store a set of *Tx* to be mined) according to the sender of an individual *Tx*. Each shard mines a block for each height and persists it in the DL [1]. There are various ways how a BC may implement sharding [17] (*cf.* Sec. II), but in most cases inter-sharding state transitions need to be performed between all the miners in shards. This can cause delays even in perfect networking conditions. Sharded BCs and DLs may run in fixed lengths of blocks called epochs. After each epoch, the Validators (*Vs*) are re-assigned to each shard. At the end of each epoch, a finalization procedure occurs by which the next epoch block is created and the number of new shards is determined based on the current number of *Vs* in the network [1].

A. Sharding Concerns

Key concerns experienced by sharding directly relate to the design of sharding mechanisms or are influenced by external factors, such as peer-to-peer networking. Four of the considerable sharding concerns include:

(i) **Edge cases** denote DLs being in the process of transitioning from one epoch to the next. The problem occurs when the sharded system is not yet prepared for transitioning to the new epoch and some *Vs* are placed in two different epochs. *E.g.,* for a 3 **Validator (*V*)** configuration consider the case (others exist, too), where all *Vs* mine the last block before the epoch block. At this moment, *V1* to *V3* are leaders in Shards S1 to S3, respectively and *V2* is in S2. After mining the last block and the assumption that *V1* and *V2* receive the state transitions from *V3*, they continue with mining epoch block, while *V3* is waiting for the state transition from *V2* due to an unstable connection. *V1* fulfills the PoS condition and can insert the epoch block, which is received by all *Vs*. Thus, *V1* is now in S3, *V2* in S1, and *V3* in S2. This new *V* assignment confuses *V3*, since he already processed the transition from S1, which will not be requested again, since with a shard ID 2, no transitions from S2 are requested. Since the shard ID 3 is now associated with *V1* and if *V1* receives a request for a state transition of S3, it will answer the request with its own state transition produced before the epoch block for S1. *V3*, however, already holds this transition and it will keep requesting it, but will never receive the expected transition. At this moment the sharded DL cannot be recovered anymore, because *V3* cannot request the transition from the right miner.

(ii) **Epoch block finality** defines epoch blocks to be final and unchangeable after creation. Since rollbacks for epoch blocks are by definition not possible, the last epoch block arriving can be accepted as valid in all cases. To ensure that all *Vs* eventually accept the same epoch block, a timer can be employed by which, after the epoch block's creation, *Vs* have to wait for a predefined time (*e.g.,* 5 s) before continuing the mining process. Thus, every *V* accepts the last epoch block and when *Vs* start mining, all hold the same epoch block that they attach their first block of the new epoch to. However, this is not always achievable!

In a 2 shards and 2 *Vs* case consider that *V1* (leader in S1) and *V2* (leader in S2) mine the last block before the epoch block. Both succeed in mining the block and *V1* receives the state transition from *V2*. Assuming that the connection between these *Vs* is now interrupted, *V* 1 can now start mining the epoch block, broadcasts it into the network, and continues mining the block after the epoch block. However, *V2* still waits for the state transition from *V1*. If *V2* now receives the transition from *V1* — be it through a rebroadcast from other nodes or by a short re-instantiation of the connection between *V1* and *V2* — *V2* is unaware of *V1* producing the epoch block and mines an epoch block himself. At this moment the BC cannot be recovered anymore, because epoch blocks always have to be final. When the connection is reestablished again, *V2* will broadcast the epoch block to *V1* and the new epoch block along with its *V*-shard assignment will be taken over by *V1* after it mined the first block (after the previous epoch block). This means that one *V* can have two different shard IDs within one epoch, which makes the entire transition and request structure fail.

(iii) **Networking effects** pose a major challenge in operating a DL, since missing rebroadcasts of lost transactions in an unstable network does impact inter-miner communications heavily. Thus, the inter-/intra-shard *Tx* transmission throughput may encounter severe performance drops. Each disconnection leads to a new request or rebroadcast, which causes additional network load and reduces the scalability of the sharded DL.

(iv) **Inter-shard communications validity** is required to ensure that the DL can be trusted. Thus, state transitions used in a sharded DL are key and the global state of the DL has to be maintained by inter-shard communications to validate *Tx*'s reliably. By letting all shards validate all transactions, the purpose of a sharding mechanism would be defeated! Thus, protocols used for inter-shard communications for state transitions may only implement a naive verification mechanism. For each state transition the shard ID and the block height of the block, where it originated from, has to be passed to other shards for verification. Therefore, malicious shards or even outsiders could create state transitions with another shard's name, if is no identity check would be performed.

B. Mechanisms for Reliable Sharding

To overcome these sharding concerns as discussed above, sophisticated approaches have to be considered to avoid *Edge cases*, while ensuring the *block finality*, laid over secure

communications. Moreover, strict and precise verification of previously mined blocks and *Tx*s shall not be neglected. It is crucial, too, to consider unstable networks, where communication delays and disconnections cause synchronization problems between all entities involved in the consensus mechanism. Otherwise, the DL will encounter *Tx* losses with time consuming or even no proper recovery method at hand. Above all, the consensus mechanism implemented has a direct impact on the overall scalability of DLs as well.

1) ***Related Work****:* In order to offer a reliable and secure sharding mechanism, different proposals employ proprietary measures by the combination of sharding and respective security features. For instance, in Bazo, in order to make sure that all miners are updated with all validated *Tx*s, even about the *Tx*s which don't belong to their shard, a protocol is used that at a certain epoch height *EH*, epoch block is inserted, denoted as eb^{EH} in Fig. 2, which consolidates the global state of the DL and re-assigns the *validators (Vs)* to the shards in a randomized fashion. Then, Shard_i with shard ID *i* produces the block $sb_i{}^h$ at height *h*. Using a proprietary synchronization mechanism, at each height *h*, all shards synchronize the global state with each other in order to have the global state before mining the next block with height *h+1* [1].

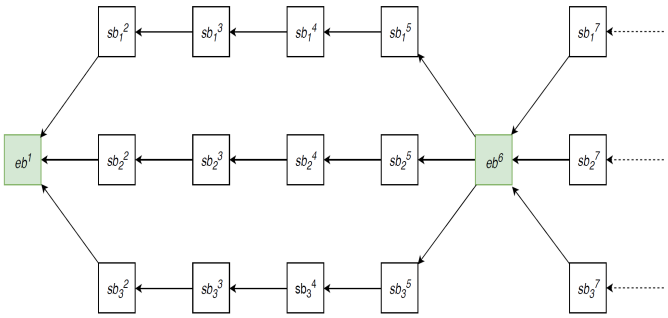

Fig. 2. Epoch Block Representation by [1]

Zilliqa is one of the BCs which have implemented the *sharding* feature [8]. Zilliqa's performance increases whenever 600 additional nodes join the network as it assigns roughly this number of nodes in each shard. The performance increase is almost linear in the number of shards, until around 1 million nodes are active in the system. A testbed with 1800 active nodes manages to reach 1218 Transactions per Second (TPS) of July 2020 [8]. Zilliqa implements a hybrid consensus mechanism. For authentication of the nodes, it implements a Proof-of-Work (PoW) mechanism. Inside the shards, however, Zilliqa employs a protocol based on Byzantine Fault Tolerance (BFT). The entire BC is managed by a committee. At the beginning of each epoch, a set of nodes is elected to participate in the committee, which assigns incoming *Tx* to shards. Transactions are sharded based on the address of the sender [8].

RapidChain proposes a sharding based BC [18]. RapidChain creates multiple committees, which are similar to shards. During the bootstrap phase, a reference committee is elected which is in charge of partitioning the set of nodes in the BC into separate committees. Each node stores $1/k$ of the BC, where k denotes the total amount of committees in the system. The committees have to be reconfigured after each epoch. In order to perform the reconfiguration of the committees as efficient as possible, and to easily add new nodes to the BC, an adapted version of Harmony's Cuckoo Rule was implemented [18] [10]. It is claimed that using the methods proposed by RapidChain and running 4000 nodes, a TPS of 7380 could be reached [18].

QuarkChain proposes a two-layered approach. One layer consists of the elastic sharded BC. Elastic in this context means that the number of shards and nodes in the BC can vary. The second layer consists of the root BC which is in charge of confirming blocks from the shards [6]. The key challenge is to have a computationally strong enough root layer in order not to bottleneck the first layer. The highest TPS that could be reached using Python was 55'039.58 TPS [16].

OmniLedger is a sharding-based BC and uses the UTXO model to process *Tx*s [9]. In order to combat sybil attacks, it operates an identity BC which runs in parallel to the actual BC where the *Tx*s are processed [9]. It utilizes state blocks as stable checkpoints which summarize the entire state of the BC and for the assignment of *Vs* to shards [9]. The shards in OmniLedger are called *Optimistic Vs* because they quickly validate *Tx* and put them into a block, creating a commitment, which with a high probabilistic likelihood won't change. Those blocks will then be checked again by the *Core Vs*. Once accepted by the *Core Vs*, the blocks produced by the *Optimistic Vs* are final. The *Core Vs* process blocks in parallel in order to maximize the scalability of the system. OmniLedger is able to reach 4000 TPS [9].

2) ***Discussion****:* The study on the related work shows that DL designs offer a subset of entities with dedicated responsibilities. However, it can be noticed that an attempt to reduce or remove the inter-shard communications is missing in the state of art related work. This applies to all other entities (used for validation or establishing a consensus) used within the introduced DLs, as well. These interesting approaches are limited in some facets such as lacking a proper *Tx* re-validation mechanism such as in Bazo [1], or they are not using any techniques to reduce the amount of the data stored in the DLs especially for IoT data. Moreover, there has been no precocious techniques applied in the related work to proactively consider the networking instability all of which will not allow the DLs to achieve a high scalability.

Another missing element is the storage optimizations *e.g.,* with *Tx* aggregation, integrated with the sharding approaches to limit the BC size growth. Especially for the IoT use cases where a large number of data transactions flow towards a BC.

III. DLIT Design and Implementation

To develop a scalable sharding mechanism for IoT use cases, DLIT is designed as a DL for small-to-medium size networks with the goal of reducing inter-shard communications for higher scalability. Comparable to related work, DLIT is designed in two layers, *i.e.,* (*i*) **Committee (CM)**

and (*ii*) **Validators (*Vs*)**. The novelty of the DLIT design includes roles, which moderate the power entities can obtain. While reducing the inter-shard communication dependency of the entire DL, DLIT materializes a highly reliable validation processes monitoring the integrity of the full DL.

The DLIT CM is an entity, where its members do not participate in the validation process performed within shards. For an efficient *Tx* management, the CM assigns *Tx*s to *Vs* in shards and performs necessary validation and authentication steps to make DLIT a trustworthy DL. *Vs* are provided with *Tx*s, which they need to validate at a particular step in the DL. Thus, DLIT *Vs* do not have their own *Tx* database, and they do not communicate with other shard *Vs*. Therefore, less redundancy at the shards level is reached. Moreover, *Tx* rebroadcasts are not necessary anymore, since *Vs* are already provided with the *Tx* they need. Thus, a global view of *Tx*s inside shards is obsolete. DLIT *Vs* no longer have to fetch, store, and delete *Tx* that they never validated earlier.

Two CM entity types exist: *CM members* and *leaders*. A leader exists in shards, too, who is the leader for one epoch being elected by the CM in a randomized leader assignment mechanism of the previous epoch. The CM leader is responsible for assigning *Tx* to shards and also to run a BFT CM inside the CM to come to an agreement, if malicious behavior is detected in DLIT. Meanwhile, all CM members validate the DL. In order to restrict the CM leader's power of a certain height, the CM leader of epoch n is responsible for the BFT of epoch $n-1$. If a user joins the *V* pool, he automatically becomes a *V*, with the chance of becoming the leader in every following epoch. If the user joins the CM pool, he automatically becomes a CM member, with the chance of becoming the leader in every following epoch, too.

A. IoT Data Transactions

Since the main focus of DLIT is preserving IoT data, a specific data *Tx* is dedicated for this purpose. The data *Tx* (DataTx) can be used to send data from one address to another and store this *Tx* inside the DL. DataTx follows the same principles as other *Tx* types in DLIT, such as the funds *Tx*. DataTx have to be signed by initiators *i.e.,* clients, using the Elliptic Curve Digital Signature (ECDSA) algorithm. The CM distributes the DataTXs among all the shards.

1) Transaction Aggregation: For scalability reasons DLIT extends the *Tx* aggregation approach of [14] to DataTx. DLIT enables aggregation of any amount of DataTxs to an aggregated DataTx based on the sender/receiver address within the limits of a block size. The main benefit of this process is to save memory in the DL by requiring to store only one aggregated *Tx* instead of all *Tx*s individually.

2) Transaction Management & Assignment: Since *Vs* do not have a global view of all *Tx*s anymore, they rely on the CM to supply them with a portion of the total *Tx* pool with *Tx*s that have not been validated yet. As shown in Fig. 4-a, at the beginning of each epoch, the CM evaluates the number of *Vs* in the system and assigns to each of them a partition of the pool of open *Tx*s according to the public address of the *Tx*s' sender. In order to do that, a new data type *Tx* assignment is introduced in DLIT. It includes all *Tx*s from the global *Tx* pool that have been assigned to a specific *V* at the time of the creation of this assignment. Shifting the responsibility for the *Tx* sharding to the CM avoids multiple executions of the same task in each shard and effectively makes *Vs* light-weight.

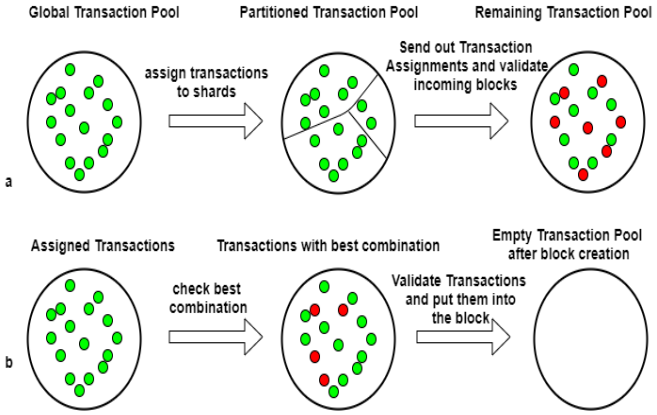

Fig. 3. Transaction Assignment and Validation in DLIT

After a block is mined, it contains a set of *Tx*s according to the *Tx* aggregation mechanism. The mined block is broadcast to the network to be validated by CM members. Moreover, the state transition is sent out to the entire network for synchronization and validation purposes. When CM receive the mined block, removed the validated *Tx*s from the open pool (red circles) and re-assign the remaining *Tx*s (the green circles) to the miners in shards. DLIT's *Tx* assignment and management mechanism function at a *V* fills up the local mempool of the *V* with *Tx*s, once the assignment was performed. The *V* chooses the optimal combination of *Tx*s to be added to the block (green circles). After the block's creation, both the *Tx*s that were validated and the ones that were not validated are deleted from the *V*'s mempool (cf. Fig. 4-b).

The CM, upon reception of the block from *Vs*, fetches all *Tx*s, which were included in the block and writes them to the closed *Tx*s pool. The same set of *Tx*s is also deleted from the open *Tx* pool. *E.g.*, in a 3 shard - 1 CM configuration, first, the global open *Tx* pool is loaded into the memory of the CM leader. In turn, the entire pool is divided into 3 different partitions, each representing one shard. Those partitions are sent by the CM leader to *Vs* assigned to each shard. The *Tx*s validated inside the block are deleted from the mempool. All remaining *Tx*s are sent out again to all *Vs* assigned. By keeping track of all *Tx*s, which were not validated by *Vs*, the CM does not have to force *Vs* to validate all *Tx*s that were assigned. *Tx*s not validated will be taken into the *Tx* assignment mechanism at the next block height again.

B. Communication and Synchronization

DLIT keeps the communication networking overhead as low as possible. Thus, a light-weight synchronization mechanism reduces inter-shard communications. Moreover, the communication between the CM and *Vs* has been designed to be

minimal. DLIT requires the *V*-CM synchronization to keep a global view of open *Tx*s and to decide, which *V* validates which *Tx*s at a certain height. Without synchronization, malicious behavior, such as double spending, could be possible.

1) Inter-Shard Communication and Synchronization: DLIT could distribute the task of aggregating state transitions to a global state to all shards, thus, the same task is performed by all *Vs*. Also, all state transitions have to be sent to all other *Vs*. If the total amount of *Vs* is denoted as n, the basic sharding mechanism leads to a total of $\frac{n*(n-1)}{2}$ connections that have to be active. Moreover, a total of $n*(n-1)$ messages have to be sent inside the network at each block height. This leads to a complexity of $\mathcal{O}(n^2)$ messages in the system at each block height, which DLIT avoids.

DLIT solves this problem by fixing the epoch length to one. Thus, state transitions are created and distributed by all *Vs*, and they are applied as well as aggregated by the leader before creating the epoch block. The last step of deleting *Tx*s that were validated by other *Vs* is omitted, because the leader does not see those *Tx*s in the mempool. Note that since only the leader *V* has to receive state transitions, the size of messages for each block height is reduced to $\mathcal{O}(n)$. This mechanism guarantees that the shard leader is up to date after receiving all state transitions. Exactly for this reason, it is the leader's task to create the epoch block with the updated global state in it, such that other shards and the CM synchronize the global state as well, along with the shard and CM leader assignment

DLIT only requires all shards to communicate with shard 1. In comparison to a regular sharding mechanism, where all miners need to send their state to each other, DLIT requires a reduced amount of communications. Fig. 4 illustrates the communication between 4 shards in DLIT. Dashed lines indicate inter-shard communications, which are not required by DLIT. Clearly ,fewer communications enable a decreased probability of packet loss and faster synchronization of miners. Hence, DLIT is less affected by unstable networks.

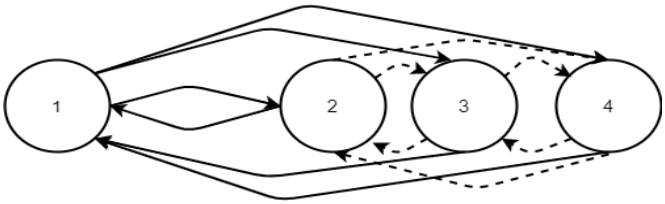

Fig. 4. Inter-shard Synchronization in DLIT

2) Inter-Committee Communication and Synchronization: Just like for shards, CM members need to synchronize with each other, since it is essential that they work on identical data to keep their mempools synchronized. It should be irrelevant for shards how many members the CM has, because the CM acts as a synchronized and unified entity. The inter-CM communication designed enables the CM to decentralize power by running the validation task not only in one node, but in all CM nodes in parallel, thus, coming to a unified and definitive decision in the end. To achieve that, all CM members receive the *Tx* assignment from the CM leader, such that every member has identical data to work with while validating. During the validation process, each CM node performs identical steps using identical data, which will, in the end, help them find a consensus of which nodes acted maliciously. This is reached, since CM member send out a CM check, containing information about which nodes they identified as being malicious. Since only the leader has to process those messages and like for inter-shard communications, it is a major concern to keep the total amount of required messages for inter-CM communications as low as possible. Thus, only the leader sends out the *Tx* assignment to other members, and only the leader receives CM check messages from other members, leading to a complexity of $\mathcal{O}(n)$ of sent messages, as opposed to $\mathcal{O}(n^2)$, with n denoting the number of nodes in the system.

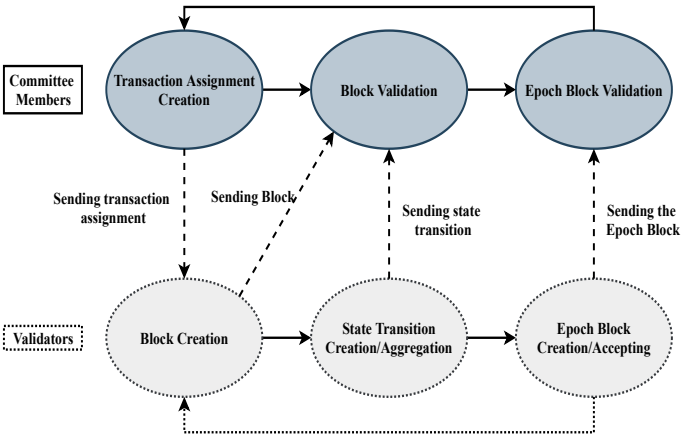

Fig. 5. DLIT's Validator-Committee Communication and Synchronization

3) Validator-Committee Communication and Synchronization: Communications between the CM and *Vs* is grouped into two types: *(i)* sending out *Tx* assignments and *(ii)* validating blocks and the epoch block. At each block height, *Vs* have to wait for the CM's *Tx* assignment. The reception of this *Tx* assignment is the starting point for mining a new block. Once the CM received and validated all blocks, it can build the new global open *Tx* pool with remaining, not validated *Tx*s. After the reception of the epoch block with its *V*'s shard assignment, it is ready to partition the global open *Tx* pool according to these *Vs* again. An epoch block with height h can only arrive at the CM, if all blocks in the epoch with height h have been mined and after all *Tx* assignments have been received.

The synchronization happening at these communications guarantees that *Vs* and the CM are always at the same block height. Figure 5 shows a simplified state machine for DLIT's CM-*V* communications. A transition to the next state is depicted by a solid arrow, while messages are represented by dashed arrows. Upon incoming messages, the machine halts until all required incoming messages have arrived. Effectively, those communications at both state machines enables them to be synchronized. *E.g.*, by construction it is not possible to send two consecutive *Tx* assignments before a block for the first *Tx* assignment has been mined and sent to the network.

Finally, the *V*-CM communication is used for updating the closed mempool. Since all CM members validate all blocks, they can write *Tx*s, which were contained in blocks to the closed mempool. Furthermore, to synchronize the global state of *Vs* with the global state of the CM and to communicate the new shard and CM leader assignments, all CM members also validate epoch blocks.

C. Authentication and Validation

While the authentication ensures that commitment keys are checked, validation ensures that DLIT's *Tx*s follow the rules.

1) ***Authentication Strategy****:* DLIT state transitions, just like blocks and epoch blocks, have to be signed using the *Vs*' private commitment key via an RSA signature mechanism. Using the public commitment key, publicly available in the DL state, the validity of the key can be checked. In a similar fashion, inter-CM communications and the shard-CM communication is equipped with signatures and checks, such that no malicious node inside or outside DLIT can assume a wrong identity. Therefore, the authenticity concerns of communications are settled.

2) ***Validation Strategy****:* The CM checks *(i)* whether *Vs* did include *Tx*s in their block that were not assigned to them and *(ii)* if state transitions that they produced correspond to actually validated *Tx*s. A DLIT CM creates its own state transition based on *Tx*s that were put inside the block and compares it to the state transition, which was produced by the shard. The check for validity of this state transition is crucial, since without checks any number of *Tx*s could be included in the relative state, remaining undetected. This is because the block creation and state transition creation processes are decoupled. Therefore, even if the block produced is valid, it is possible that the state transition contains wrong information.

The task of the first DLIT shard, called the *shard leader*, is to aggregate all state transitions with its own state and to consolidate it in the epoch block. In turn, all other nodes in the network accept the global state from that epoch block. The validity (and authenticity) of the epoch block is, therefore, crucial within a trustworthy system. Since authenticity was already addressed by previous implementations, the validity check in DLIT are performed by CM via aggregating all relative states from all shards in the system, creating an aggregated relative state and comparing that relative state to the difference between the epoch block being checked and the epoch block of the previous height.

D. Implementation and Consensus Mechanism

The DLIT implementation is based on the PoS-based Bazo BC [14] and [1], for which details and source code are available at [3] and [2]. Most importantly, the DLIT consensus is twofold, including the shard and the CM side.

Consensus on shards: The shard leader in S1 operates in a special role inside the shard pool. Before being able to mine a block, the *Tx* assignment has to arrive from the CM, then a block is mined. The leader waits for all state transitions from other shards (2 ... N) and requests missing ones, if necessary. For each state transition, it updates its local state by applying the relative state. After all state transitions have been processed the leader operates on the updated current global state of the DL as the only shard node with this global information. Therefore, it produces *(a)* the epoch block containing the global state and *(b)* a new random validator shard assignment to assign a new random CM leader.

In contrast to other shards, where validators have to wait for the *Tx* assignment to mine a block and to send out the state transition, shard leader wait for the epoch block to arrive to take over the new global state along with the new shard assignment. From a shard's perspective, consensus is reached.

Consensus in Committee: At the beginning of each round, only the CM leader is active by running the BFT consensus mechanism, slashing malicious nodes of the previous round, and creating/sending *Tx* assignments. Other CM members have to wait to receive all *Tx* assignments before being able to continue.Afterwards, all CM members behave similarly, starting to listen to, requesting, and authenticating blocks and state transitions in parallel.

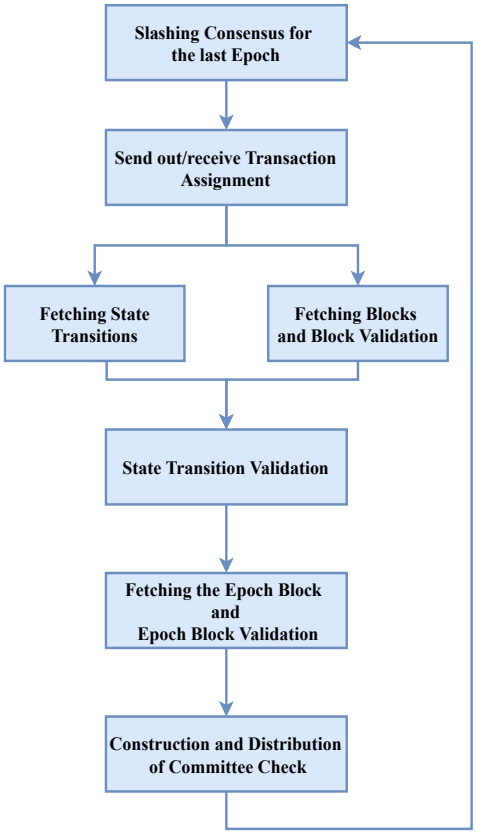

Fig. 6. Inter-Committee Communications in DLIT

This parallel execution ensures that DLIT saves time for fetching state transitions from the network. During a block validation, the validity of PoS and *Tx*s contained is checked. A relative state is constructed to validate state transitions at the next step, which will be entered once both previous steps are performed. State transitions and the epoch block are validated.

TABLE I
COMPARISON OF DLIT WITH RELATED WORK

	Zilliqa	Rapidchain	QuarkChain	Omniledger	DLIT
Use case	Decentralized Apps	Fund Transactions	General	Fund Transactions	IoT data
Consensus	Hybrid of PoW/BFT	BFT-based	Hybrid (Boson)	ByzCoinX	Hybrid of BFT/PoS
Authentication	Committee	Committee per shard	Committee	Identity BC, Core Validator	Committee
Scalability (TPS)	1,200 at 1,800 nodes	7,300 at 4,000 nodes	50,000 (local setup)	1,800 at favorable conditions with low number of adversaries	1,300 TPS in unstable networks with 3 shards
Permission	Public permissionless	Public permissionless	Public permissionless	Public permissionless	Private consortium

Next, the epoch block with its new state and assignments is accepted, and all CM members send out their CM checks naming members, who acted maliciously in that round. Those CM checks can be used by the CM leader in the next round for running the BFT consensus mechanism.

Slashing is referred to as the process to detect and fine malicious nodes. DLIT regulates slashing through the BFT consensus mechanism. Since all CM members validate the work of all validators at the end of each round, CM members prepare a summary of all nodes for which they detected malicious behavior. Thus, they create a CM check, containing all validator addresses who acted maliciously as well as all CM member addresses who acted maliciously in the epoch. The message is then signed and broadcast to the network. In the next epoch, the new CM leader collects all of those CM checks and based on the complete set of these, performs the BFT consensus mechanism. The mechanism is based on voting, which means that based on the total amount of CM members in the system, a certain amount of those has to vote to slash a specific node in the DL. Thus, if no node in the network acts maliciously, consensus is reached in shards without an active intervention by the CM. For all members of the DL, the epoch block serves as a synchronization of the global state. If malicious behavior is detected, consensus can only be reached by applying slashing, which issues fine *Tx*s.

E. IoT Data Storage Mechanism

Whenever a data *Tx* is validated, the committee takes the data from the *Tx* and stores it in DLIT. For each DLIT client account, the committee maintains the data structure called “data summary". The data summary consists of an *Address* field, *i.e.,* the sender’s address, and a *Data* field, *i.e.,* a slice of bytestreams. It has the type [][]byte and stores data that the sender sent to the network. DataTxs contains all DataTxs from the respective block, storing the IoT data in addition to the address of the sender as the key to access this slice. Thus, a global and updated view of the data in the DL is kept at all times. For higher efficiency, a map is created with the sender as the key and all sender’s DataTx are added in a slice as the respective value. This map is filled by iterating through the entire slice of DataTxs to be added. Then, the database will be iterated by sender. In each round, the database is checked, if there is already a “data summary" for a sender. If so, it will be fetched, otherwise a new “data summary" will be created. By iterating through all DataTxs from the respective sender the bytestreams contained in individual DataTxs are appended to the slice in this “data summary". After all DataTxs have been handled, the “data summary" is written back to the database.

IV. EVALUATIONS

The DLIT performance has been evaluated within a local test set-up of 60 wallets, partitioned into 8 separate partitions, each sending 3,000 *Tx*s within the shards 1-4. The block size was kept at 800 Byte and *Tx*s were sent in batches, ensuring that *Tx*s from as many different wallets as possible are available in the open mempool.

Fig. 7 overviews performance results in TPS for DLIT. It can be seen that with the different numbers of *CM members* and the number of shards adding shards increases the performance, while adding CM members indicates a higher communication overhead. Within this experiment the number of shards with a single CM member show the best performance. Also, it can be observed that the sharding mechanism shows the potential of a scalability benefit, since DLIT’s use with larger number of shards and one CM member behave well. This is due to the enhanced *transaction management*, since time-intense DL work is distributed among these shards. However, once the 3 shards threshold is reached, adding more miners does not improve the scalability further. The reason for this is the larger number of shards and CM members requires more inter-shard and inter-CM communications, causing additive network delays and security processing, which determines the current trade-off against scalability. Especially within unstable networks this situation is more visible.

If DLIT shall validate *Tx*s at the highest possible pace, a small CM needs to be determined. If security concerns prevail, a larger CM is advisable at the cost of smaller TPS. The prototypical evaluation has shown that for a configuration of 2 CM members and 3 shards a moderate centralization of the CM was reached, while benefits in performance are gained via the sharding mechanism. While DLIT ic compared with related work (*cf.* Table I), it can be concluded that DLIT is a scalable DL, which, even with a limited number of validators and only a few shards, offers a high *Tx* validation rate.

V. SUMMARY AND FUTURE WORK

This paper proposed and evaluated DLIT, a new “Distributed Ledger for IoT Data". For reaching data persistence in the

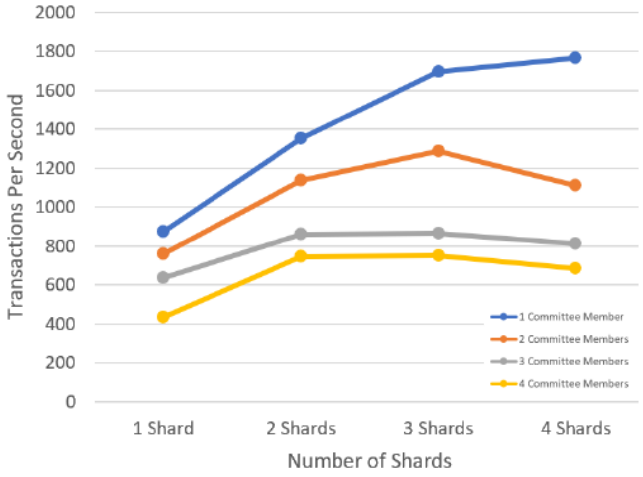

Fig. 7. DLIT Performance Results

Internet-of-Things (IoT), DLIT is based on the two consensus layers of Byzantine Fault Tolerance (BFT) and Proof-of-Stake (PoS) and maintained by two entities of Committees and Validators. DLIT reduces or even removes the need for inter-shard communications in many cases discussed. Based on the secure *Tx* block validation mechanism, DLIT implements a slashing mechanism to fine malicious nodes. Finally, DLIT divides the *Tx* assignment, validation, verification, and storage responsibility between nodes and employs a *Tx* aggregation mechanism to reach a moderate DL size growth as the first sharded DL employing such a technique for IoT data.

To reach even higher TPS rates it is planned to improve this version of a DLIT committee by applying a partial BFT consensus mechanism. Moreover, evaluations of DLIT with an increased number of nodes on the committee and shard sides are foreseen.

REFERENCES

[1] K. Aydinli, "Design and Implementation of a Scalable IoT-based Blockchain," https://files.ifi.uzh.ch/CSG/staff/Rafati/Kursat-Aydinli-MA.pdf, last visit: July 20, 2020.

[2] R. Beckmann, https://github.com/oigele/bazo-miner, last visit= July 20, 2020.

[3] ——, "Enhancing the Scalability of A Sharded Blockchain," March 2020. [Online]. Available: https://owncloud.csg.uzh.ch/index.php/s/MmAJnsYHDaRzrbS

[4] M. Beedham, "Blockchain Sharding Made So Simple Your Dog Would Understand," https://thenextweb.com/hardfork/2019/01/18/explainer-blockchain-sharding-beginners/, last visit: July 20, 2020.

[5] T. Bocek and B. Stiller, *Smart Contracts - Blockchains in the Wings*. Tiergartenstr. 17, 69121 Heidelberg, Germany: Springer, jan 2017, pp. 169–184.

[6] C. Crunch, "QuarkChain Review — A New Scalable Blockchain Looking to Dethrone Ethereum?" https://hackernoon.com/quarkchain-review-a-new-scalable-blockchain-looking-to-dethrone-ethereum-9ffc0e814772, last visit: July 24, 2020.

[7] M. Drake, "Understanding Database Sharding," https://www.digitalocean.com/community/tutorials/understanding-database-sharding, last visit: July 20, 2020.

[8] B. Garner, "What Is Zilliqa (ZIL)? | The Complete Guide to the High Throughput Blockchain," https://coincentral.com/zilliqa-beginners-guide/, last visit: July 24, 2020.

[9] E. Kokoris Kogias, P. S. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. A. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," p. 16, 2018. [Online]. Available: http://infoscience.epfl.ch/record/255586

[10] A. Kothari, "Understanding Harmony's Cuckoo Rule for Resharding," https://medium.com/harmony-one/understanding-harmonys-cuckoo-rule-for-resharding-215766f4ca50, accessed: 19.02.2019.

[11] S. R. Niya, D. Dordevic, A. G. Nabi, T. Mann, and B. Stiller, "A Platform-independent, Generic-purpose, and Blockchain-based Supply Chain Tracking," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 11–12. [Online]. Available: https://ieeexplore.ieee.org/document/8751415

[12] S. R. Niya, S. S. Jha, T. Bocek, and B. Stiller, "Design and Implementation of an Automated and Decentralized Pollution Monitoring System with Blockchains, Smart Contracts, and LoRaWAN," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/8406329

[13] S. R. Niya, E. Schiller, I. Cepilov, and B. Stiller, "BIIT: Standardization of Blockchain-based I2oT Systems in the I4 Era," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/document/9110379

[14] S. R. Niya, F. Maddaloni, T. Bocek, and B. Stiller, "Toward Scalable Blockchains with Transaction Aggregation," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 308–315.

[15] E. Schiller, S. R. Niya, T. Surbeck, and B. Stiller, "Scalable Transport Mechanisms for Blockchain IoT Applications," in *2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, 2019, pp. 34–41. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9000673

[16] Q. C. Team, "After Scaling, Why Do Public Chains Still Fail to Land?" https://medium.com/quarkchain-official/after-scaling-why-do-public-chains-still-fail-to-land-61eb74bdabe, accessed: 19.02.2019.

[17] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, "Survey: Sharding in Blockchains," *IEEE Access*, Vol. 8, pp. 14 155–14 181, 2020.

[18] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.